패턴 매칭

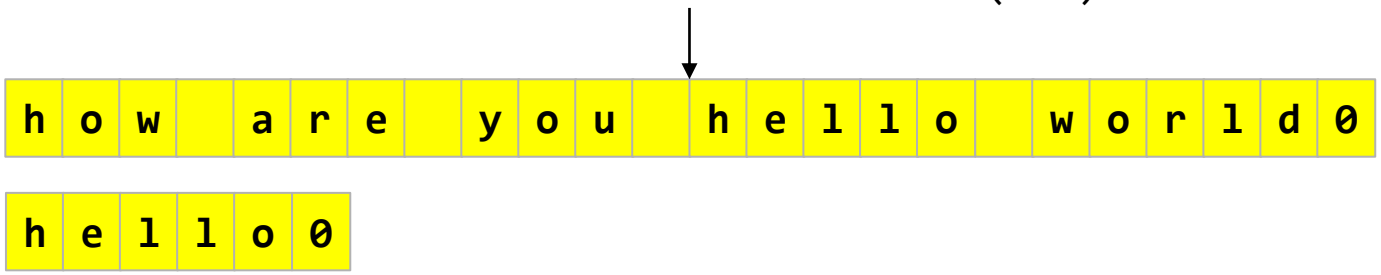
- 패턴 매칭의 기본 원리를 이해한다.
- 고급 패턴 매칭을 분석 한다.

1) 기본 패턴 매칭
2) 고급 패턴 매칭

패턴 매칭 개념

```
#include <stdio.h>
#include <string.h>
int main()
{
    char y[] = "how are you hello world";
    char x[] = "hello";

    char *p;

    p = strstr( y, x );
    printf("p=%s\n", p );
    return 0;
}
```

바깥 회전수 : (strlen(y)-strlen(x))+1 => 19
안쪽 회전수 : strlen(5)
O(n*m)



패턴 매칭 참고 자료

패턴매칭 참고 사이트

http://www-igm.univ-mlv.fr/~lecroq/string/

Brute Force algorithm

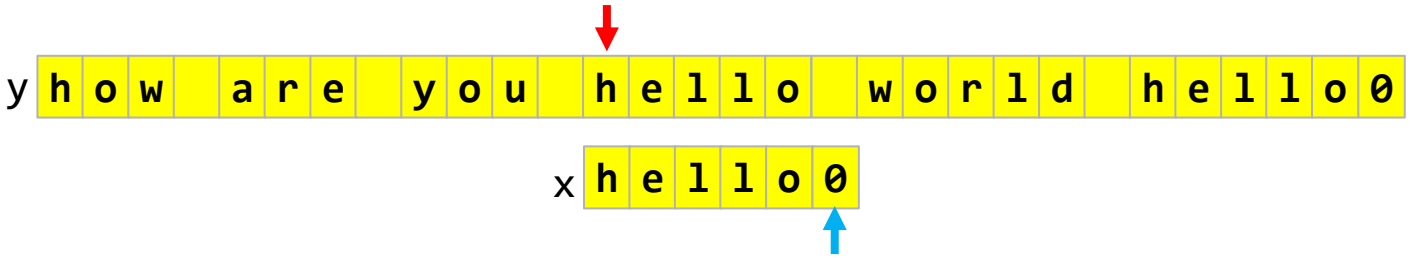
```
void BF(char *x, int m, char *y, int n) {
   int i, j;

   /* Searching */
   for (j = 0; j <= n - m; ++j) {
      for (i = 0; i < m && x[i] == y[i + j]; ++i);
      if (i >= m)
         OUTPUT(j);
   }
}
```

바깥 회전수 : n - m + 1 => 25
안쪽 회전수 : m

O(n*m)

n=29

m=5



카프 라빈 패턴 매칭 알고리즘

```
#define REHASH(a, b, h) ((((h) - (a)*d) << 1) + (b))

void KR(char *x, int m, char *y, int n) {
   int d, hx, hy, i, j;

   /* Preprocessing */
   /* computes d = 2^(m-1) with
      the left-shift operator */
   for (d = i = 1; i < m; ++i)
      d = (d<<1);

   for (hy = hx = i = 0; i < m; ++i) {
      hx = ((hx<<1) + x[i]);
      hy = ((hy<<1) + y[i]);
   }

   /* Searching */
   j = 0;
   while (j <= n-m) {
      if (hx == hy && memcmp(x, y + j, m) == 0)
         OUTPUT(j);
      hy = REHASH(y[j], y[j + m], hy);
      ++j;
   }

}
```

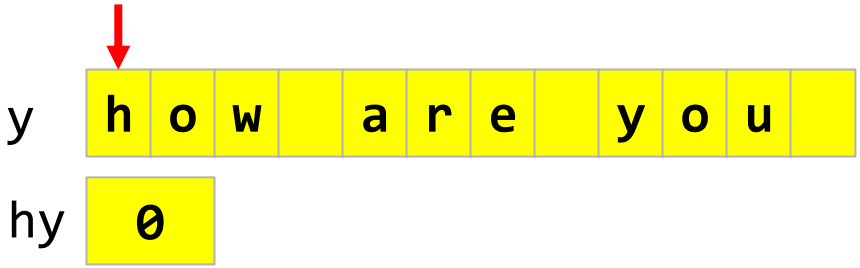
hx	0					
x	h	e	l	l	o	0
	h	e	l	o	l	0

```
hx = 0*2 + 'h';
hx = 'h'*2 + 'e'
hx = 'h'*2^2 + 'e'*2^1 + 'l'*2^0
hx = 'h'*2^3 + 'e'*2^2 + 'l'*2^1 +'l'*2^0
hx = 'h'*2^4 + 'e'*2^3 + 'l'*2^2 +'l'*2^1 +'o'

1234 => 1*10^3 + 2*10^2 + 3*10^1 + 4*10^1
1243 => 1*10^3 + 2*10^2 + 4*10^1 + 3*10^1
```

카프 라빈 패턴 매칭 알고리즘

```
#define REHASH(a, b, h) ((((h) - (a)*d) << 1) + (b))
```

```
hy  = 'h'*2^4 + 'o'*2^3 + 'w'*2^2 +' '*2^1 +'a'
hy' = 'o'*2^4 + 'w'*2^3 + ' '*2^2 +'a'*2^1 +'r'
hy' = ((hy - y[j]*2^4)<<1) + y[j+m]
```



Shift Or algorithm

```
int preSo(char *x, int m, unsigned int S[]) {
   unsigned int j, lim;
   int i;
   for (i = 0; i < ASIZE; ++i)
     S[i] = ~0;
   for (lim = i = 0, j = 1; i < m; ++i, j <<= 1) {
     S[x[i]] &= ~j;
     lim |= j;
   }
   lim = ~(lim>>1);
   return(lim);
 }

 void SO(char *x, int m, char *y, int n) {
   unsigned int lim, state;
   unsigned int S[ASIZE];
   int j;
   if (m > WORD)
     error("SO: Use pattern size <= word size");

   /* Preprocessing */
   lim = preSo(x, m, S);

   /* Searching */
   for (state = ~0, j = 0; j < n; ++j) {
     state = (state<<1) | S[y[j]];
     if (state < lim)
       OUTPUT(j - m + 1);
   }
 }
```

Shift Or algorithm

```
int preSo(char *x, int m, unsigned int S[]) {
   unsigned int j, lim;
   int i;
   for (i = 0; i < ASIZE; ++i)
     S[i] = ~0;
   for (lim = i = 0, j = 1; i < m; ++i, j <<= 1) {
     S[x[i]] &= ~j;
     lim |= j;
   }
   lim = ~(lim>>1);
   return(lim);
 }
```

x	h	e	l	l	o	0		
S['h']	1	1	1	1	1	1	1	0
S['e']	1	1	1	1	1	1	0	1
S['l']	1	1	1	1	0	0	1	1
S['o']	1	1	1	0	1	1	1	1
lim	1	1	1	1	0	0	0	0

Shift Or algorithm

```
void SO(char *x, int m, char *y, int n) {
   unsigned int lim, state;
   unsigned int S[ASIZE];
   int j;
   if (m > WORD)
     error("SO: Use pattern size <= word size");

   /* Preprocessing */
   lim = preSo(x, m, S);

   /* Searching */
   for (state = ~0, j = 0; j < n; ++j) {
     state = (state<<1) | S[y[j]];
     if (state < lim)
       OUTPUT(j - m + 1);
   }
 }
```

y	h	e	l	l	o	0

S['h']	1	1	1	1	1	1	1	0
S['e']	1	1	1	1	1	1	0	1
S['l']	1	1	1	1	0	0	1	1
S['o']	1	1	1	0	1	1	1	1

	1	1	1	0	1	1	1	1
state<<1	1	1	1	0	1	1	1	0
state	1	1	1	0	1	1	1	1
lim	1	1	1	1	0	0	0	0

MP 알고리즘 분석

```
void preMp(char *x, int m, int mpNext[]) {
   int i, j;

   i = 0;
   j = mpNext[0] = -1;
   while (i < m) {
      while (j > -1 && x[i] != x[j])
         j = mpNext[j];
      mpNext[++i] = ++j;
   }
}
```

x	g	a	g	g	g	a	g	g	
mpNext	-1	0	0	1	1	1	2	3	4

-1 0 0 1 1 1 2 3 4

MP 알고리즘 분석

```
void preMp(char *x, int m, int mpNext[]) {
   int i, j;

   i = 0;
   j = mpNext[0] = -1;
   while (i < m) {
      while (j > -1 && x[i] != x[j])
         j = mpNext[j];
      mpNext[++i] = ++j;
   }
}
```

mpNext : 틀린 문자가 발견된 위치의 앞서 일치된 문자열속의 접두사와 접미사의 일치수를 기록한 배열

x	g	a	g	g	g	a	g	g	
mpNext	-1	0	0	1	1	1	2	3	4

y	g	c	g	g	g	a	g	g
x	g	a	g	g	g	a	g	g

y	g	a	g	g	g	c	g	g
x	g	a	g	g	g	a	g	g

y	g	a	g	g	g	a	g	c
x	g	a	g	g	g	a	g	g

y	g	a	g	g	g	a	g	g	g
x	g	a	g	g	g	a	g	g	0

MP 알고리즘 분석

```
void preMp(char *x, int m, int mpNext[]) {
   int i, j;

   i = 0;
   j = mpNext[0] = -1;
   while (i < m) {
      while (j > -1 && x[i] != x[j])
         j = mpNext[j];
      mpNext[++i] = ++j;
   }
}
```

mpNext : 틀린 문자가 발견된 위치의
앞서 일치된 문자열속의 접두사와 접미사의
일치수를 기록한 배열

x	g	a	g	g	g	a	g	g	
mpNext	-1	0	0	1	1	1	2	3	4

y	g	a	g	g	g	c	g	g
x	g	a	g	g	g	a	g	g

shift = i - mpNext[i]

MP 알고리즘 분석

```
void MP(char *x, int m, char *y, int n) {
   int i, j, mpNext[XSIZE];

   /* Preprocessing */
   preMp(x, m, mpNext);

   /* Searching */
   i = j = 0;
   while (j < n) {
      while (i > -1 && x[i] != y[j])
         i = mpNext[i];
      i++;
      j++;
      if (i >= m) {
         OUTPUT(j - i);
         i = mpNext[i];
      }
   }
}
```

mpNext	-1	0	0	1	1	1	2	3	4

y	g	a	g	g	g	a	g	c
x	g	a	g	g	g	a	g	g

shift = i - mpNext[i]

KMP 알고리즘 분석

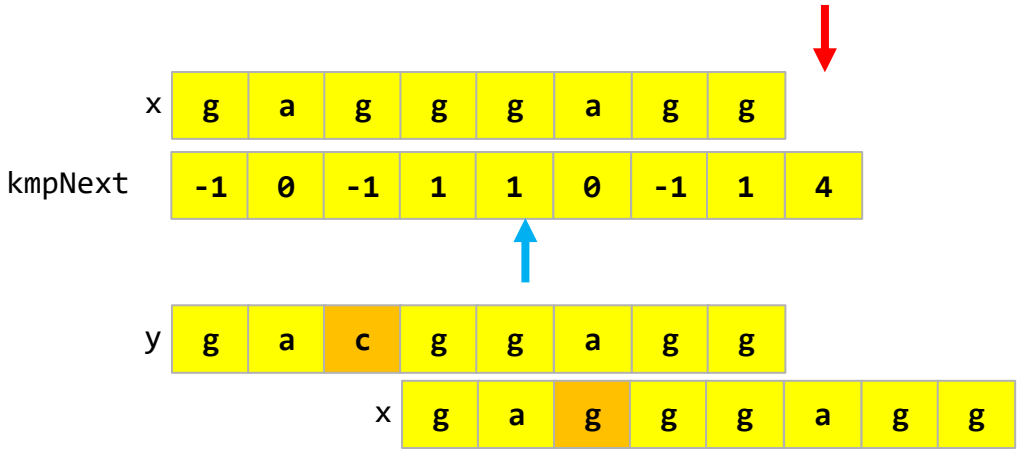
```
void preKmp(char *x, int m, int kmpNext[]) {
   int i, j;

   i = 0;
   j = kmpNext[0] = -1;
   while (i < m) {
      while (j > -1 && x[i] != x[j])
         j = kmpNext[j];
      i++;
      j++;
      if (x[i] == x[j])
         kmpNext[i] = kmpNext[j];
      else
         kmpNext[i] = j;
   }
}
```

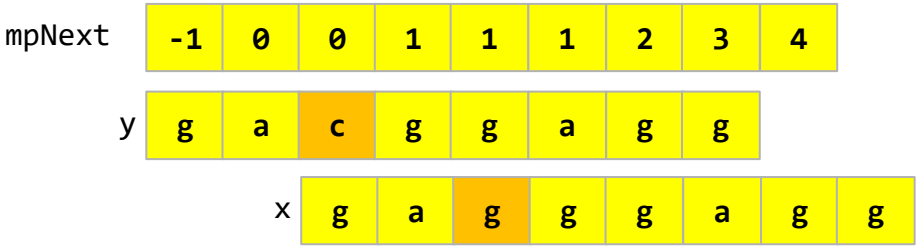
x	g	a	g	g	g	a	g	g	
kmpNext	-1	0	-1	1	1	0	-1	1	4
mpNext	-1	0	0	1	1	1	2	3	4

KMP 알고리즘 분석

```
void preKmp(char *x, int m, int kmpNext[]) {
   int i, j;

   i = 0;
   j = kmpNext[0] = -1;
   while (i < m) {
      while (j > -1 && x[i] != x[j])
         j = kmpNext[j];
      i++;
      j++;
      if (x[i] == x[j])
         kmpNext[i] = kmpNext[j];
      else
         kmpNext[i] = j;
   }
}
```

shift = i - kmpNext[i];





KMP 알고리즘 분석

```
shift = i - kmpNext[i];
```

x	g	a	g	g	g	a	g	g	
kmpNext	-1	0	-1	1	1	0	-1	1	4

y	g	a	g	g	g	a	c	g

x	g	a	g	g	g	a	g	g

mpNext	-1	0	0	1	1	1	2	3	4

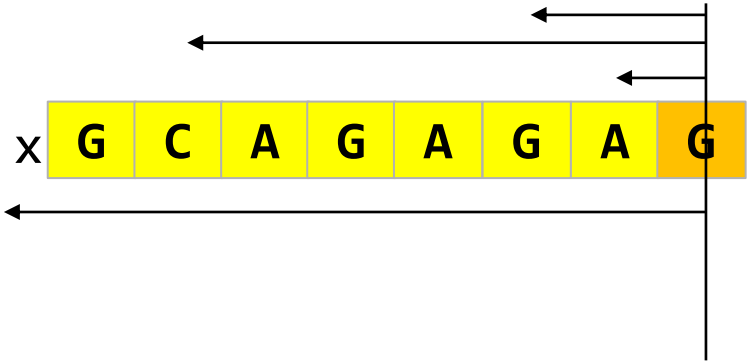
y	g	a	g	g	g	a	c	g

x	g	a	g	g	g	a	g	g

BM 알고리즘 분석

c	A	C	G	T
bmBc[i]	1	6	2	8

i	0	1	2	3	4	5	6	7
x[i]	G	C	A	G	A	G	A	G
suff[i]	1	0	0	2	0	4	0	8
bmGs[i]	7	7	7	2	7	4	7	1



y	G	C	A	T	C	G	C	A
x	G	C	A	G	A	G	A	G

Shift by: 1 (*bmGs*[7]=*bmBc*[A]-8+8)

y	G	C	A	T	C	G	C	C
x	G	C	A	G	A	G	A	G

Shift by: 1 (*bmGs*[7]=*bmBc*[C]-8+8)

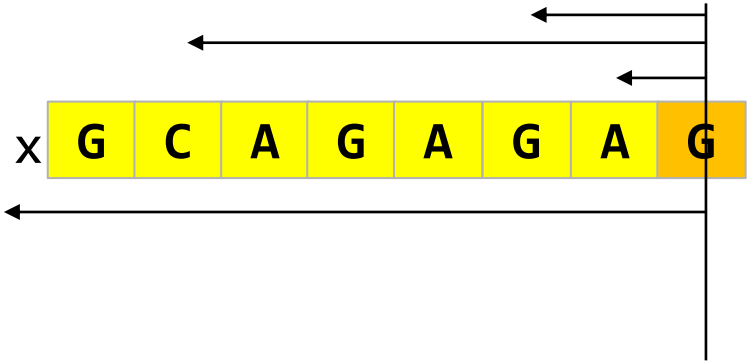
y	G	C	A	T	C	G	A	G
x	G	C	A	G	A	G	A	G

Shift by: 1 (*bmGs*[7]=*bmBc*[C]-8+5)

BM 알고리즘 분석

c	A	C	G	T
bmBc[i]	1	6	2	8

i	0	1	2	3	4	5	6	7
x[i]	G	C	A	G	A	G	A	G
suff[i]	1	0	0	2	0	4	0	8
bmGs[i]	7	7	7	2	7	4	7	1



y	G	C	A	T	C	G	C	A
x	G	C	A	G	A	G	A	G

Shift by: 1 (*bmGs*[7]=*bmBc*[A]-8+8)

y	G	C	A	T	C	A	A	G
x	G	C	A	G	A	G	A	G

Shift by: 1 (*bmGs*[7]=*bmBc*[A]-8+6)

BM 알고리즘 분석

c	A	C	G	T
bmBc[i]	1	6	2	8

i	0	1	2	3	4	5	6	7
x[i]	G	C	A	G	A	G	A	G
suff[i]	1	0	0	2	0	4	0	8
bmGs[i]	7	7	7	2	7	4	7	1

y	G	C	A	T	C	A	A	G
x	G	C	A	G	A	G	A	G

Shift by: 1 *bmGs*[5]

BM 알고리즘 분석

G	C	A	G	A	G	A	G								
		G	C	A	G	A	G	A	G						
G	C	A	G	A	G	A	G								
				G	C	A	G	A	G	A	G				
G	C	A	G	A	G	A	G								
						G	C	A	G	A	G	A	G		
G	C	A	G	A	G	A	G								
								G	C	A	G	A	G	A	G
G	C	A	G	A	G	A	G								
										G	C	A	G	A	G
G	C	A	G	A	G	A	G								
												G	C	A	G
G	C	A	G	A	G	A	G								
														G	C
G	C	A	G	A	G	A	G								
															G

BM 알고리즘 분석

c	A	C	G	T
bmBc[i]	1	6	2	8

i	0	1	2	3	4	5	6	7
x[i]	G	C	A	G	A	G	A	G
suff[i]	1	0	0	2	0	4	0	8
bmGs[i]	7	7	7	2	7	4	7	1

y	G	C	A	T	C	C	A	G
x	G	C	A	G	A	G	A	G

Shift by: 4 max(*bmGs*[5] , bmBc[C]-8+6)

y	G	C	A	T	C	G	A	G
x	G	C	A	G	A	G	A	G

Shift by: 4 max(*bmGs*[5] , bmBc[G]-8+6)

y	G	C	A	T	C	T	A	G
x	G	C	A	G	A	G	A	G

Shift by: 6 max(*bmGs*[5] , bmBc[T]-8+6)